# Chapter 5

# Object Oriented Programming

We call it as ‘Object Oriented Programming(OOP), but, in real sense it is ‘Class oriented programming’. JAVA is a package of many classes and methods. In our program we reuse these classes and methods as per our requirements. Class is not used directly, first of all we create an object from the class and through that object we access all methods and parameters of that class. An object is the instance of a class, a real thing, through which we can work. From a class we can create multiple number of objects. All classes are stored within different packages and all packages are under java package. Learning of JAVA programming means- we should learn about some classes and methods available under those classes. Example of some classes used in this book are:

**String**: to do any string operations
**DataInputString**: to input string type of data from input device
**Scanner**: to input any number or string from keyboard or input device.
**Graphics**: to do all graphical operations like: mouse, icon, click, create picture etc.
**Date**: to do any manipulation over date type of data
**Math**: to do mathematical operations
**Character**: to do different international character codes.

Generally, in programming, we process some data and deliver some required information. A program does three things- read data, process data and output data. Without these three things there are nothing in concept of programming. Some programs work internally and we can’t see anything externally, as example, when we make internet connection through MODEM(Modulation and DeModulation), Dangle-we see nothing happened, but the software present inside the computer has done lot of programming works and made connection with the server computer of ISP(Internet Service Provider). So, if the actions of program are not visible also, which may act on internal devices, do some changes- can perform desired jobs; all of these are changing the status of a device from one mode to another mode. These are the examples of different processes. It is said here because, object is some material substances, we understand that. We process data of different objects in computer and display information on monitor, store into the Disk or Pendrive etc. But, a process, which is not a material substance, can be an object also which we can’t touch but feel. Examples of processes are: connection, write data, read data, start rocket, monitor actions, create log file etc. All of these jobs are part of process and we create object and through that we perform these jobs in programming.

Think about we process data of some material subjects like: book, school, student, teacher, tree, animal, cow, home etc. Sometimes, verbs are used as nouns and can be used as objects, as example: hope, desire, dream etc. Suppose, we want to make a game on different types of dreams seen by different patients to analyse the brain functions of a parent. Then, properties or attributes of Dream should be collected as: time of dream, duration, patient name, type of disease etc. So, we conclude discussion of object as: all materials are objects and all processes are also objects.

## Creating an Object:

```
    String st;  // creating object st
   st = new String(); //initiating st or assigning space as class String.
```
Or,
```
    Srting st = new String();
   String st2= new String(); or ,  String st3=st2;
```

Here st, st2 are objects and String() is the constructor of the Class **String**.

```
DataInputString in, in2, in3;
Scanner kb, mp, camera;
```

In this way multiple objects can be declared at a time.

**Problem:** wap to create a class Student with variables ‘name, city and age’ along with a method ‘printAll()’ to display all data. Create another main class by name ‘Stest’ of main(), create two objects s1 and s2 of students and access all properties of Student class to print all data.

```
//Stest.java: Write a program to create a class Student with data ‘name, city and age’ along
//with method printAll
//to display the data. Create the two objects s1 and s2 to declare and access the values.
class Student
{
            String name, city;
            int age;
            static int m;
            void printAll()
            {
                        System.out.println(" Name : "+name);
                        System.out.println("City : "+city);
                        System.out.println("Age : "+age);
            }
}
class Stest
{
            public static void main(String args[])
            {
                        Student s1=new Student();
                        Student s2=new Student();
                        s1.name="Komal"; s1.city="Gujrat";
                        s1.age=26;
                        s2.name="Monika"; s2.city="Kolkata";
                        s2.age=22;
                        s2.printAll(); s1.printAll();
                        s1.m=20; System.out.println("(1) s1.m = "+s1.m);
                        s2.m=25;
                        Student.m=22;
                        System.out.println("(2) s1.m = "+s1.m);
                        System.out.println("(3) s2.m = "+s2.m);
                        System.out.println("(4) Student.m ="+Student.m);
            }
}
```

```
OUTPUT:
Name : Monika
City : Kolkata
Age : 22
 Name : Komal
City : Gujrat
Age : 26
(1) s1.m = 20
(2) s1.m = 22
(3) s2.m = 22
(4) Student.m =22
```

**Discussion:** in this example how to create a class and an object from that class is demonstrated. Here, two objects, s1 and s2, are created for two student's particulars. Objects like s1 and s2 are initiated with memory block by the instruction as '*Student s1=new Student()*' ; object *s1* is created by first command of left-side, and the commands of right-side created a memory block as required by the object *s1*, memory block is also linked with *s1*. Here, *Student()* is the default constructor. When we declare static as a variable, that is a unique variable name and will not be created within multiple objects but the value of all samples are same. Value of static variable will not be changed. Here, if we observe within this example at last four lines, when we change value of *m*, it reflects changes within all samples of *m* of all objects. It proofs that value of *m* is only one value as it is *static*. But, when we assigned different values within different samples of m it did not make any compilation error. As shown below, there are three values are assigned within s1.m, s2.m and Student.m, but when we print we get the last assigned value for all three samples:

```
s2.age=22;
                    //s2.printAll();
                    //s1.printAll();
s1.m=20;             //System.out.println("(1) s1.m = "+s1.m);
s2.m=25;
Student.m=22;
```

## Reuse of Class file:

It is the best practice of programming to design and write all code as module wise. It is called modular designing. Now-a-days, all designing are of module based design. As an example, in building of apartment, modular and sliding windows can be used in any flat-apartment. An engine is an assembled thing of many small modules, a motor cycle is an assembled bike, a bridge is made assembling some small modules, a software is developed wiring small units. This is the basic concept of modern structured engineering.

JAVA is a fully class based structured programming language. The modules created now, can be reused in other programs. In programming, using of some module or portion of coding to other program is known as reuses of program. Here, an example is shown where a class "*Doctor.class*" is created and is reused within other program of "*UseDoctor.java*" which will get access of all properties of the *Doctor.class* module. The both programs are shown below:

**Problem:** wap to create a class by name '*Doctor.java*' having some methods and variable, make a class file as *Doctor.class*. Then create another program '*UseDoctor.java*' which will call the *Doctor.class* and access methods of that class.

```
//Doctor.java: only a class is created, but, there is no main() method.
public class Doctor
{
        private int patient;
        public int getPatient()
        {
                return patient;
        }
        public void setPatient(int  numPatient)
        {
                patient = numPatient;
        }
        public void CallDoctor()
        {
                System.out.println("Please wait, Doctor is coming:");
        }
}
```

This program 'Doctor.java' is compiled and 'Doctor.class' is generated. Let us see the second program.

```
//UseDoctor.java: this program uses the previously created and compiled 'Doctor.class' file
import java.util.*;
public class UseDoctor
{
        public static void main(String[] args)
        {
                int number;
                Doctor aDoctor = new Doctor();
                Scanner keyboard = new Scanner(System.in);
                System.out.print("Enter number of patients waiting for the Doctor  " );
                number = keyboard.nextInt();
```

```
                    aDoctor.setPatient(number);
                    System.out.println("Patients: " + aDoctor.getPatient()+ " are waiting for
                                                                                        Doctor");
                    aDoctor.CallDoctor();
          }
}
```

```
OUTPUT:
E:\example>java UseDoctor
Enter number of patients waiting for the Doctor  15
Patients: 15 are waiting for Doctor
Please wait, Doctor is coming:
```

**Discussion:** In this program, 'UseDoctor.java' uses 'Doctor.class' which is residing in the default directory and used in this way:

```
Doctor aDoctor = new Doctor();
```

This class creates an object by name 'aDoctor' and also initiated. It will search the class within the present program, if not fount, then search within the hard disk, when found, it will read and load within the computer memory. Note that the 'Doctor.class' has three methods as: getPatient(), setPatient(int  numPatient) and CallDoctor(). These three methods are called from the caller program "UseDoctor.java" in this way:

```
aDoctor.setPatient(number);
```

This method 'setPatient()' was declared within the class 'Doctor.class' and accessed from the object 'aDoctor'. It should be noted that 'aDoctor' is an object but not derived Class. Always, parameters and methods of a class are accessed from an object not from a class.
But, note that the 'patient' variable is declared as *private* identifier, so, this variable is not visible within other class. To test this theory, within the 'Doctor' class we have assigned value as:

```
private int patient=10;
```

and then tried to access the value of 'patient' variable from 'UseDoctor.java' program writing the extra line at the rightmost position as 'aDoctor.patient' to verify whether the private variable of 'Doctor' can be accessed from another class or not.
When we compile the program 'UseDoctor.java', we get this error message clearly mentioning that it is declared as private. Please, check the error message:

```
UseDoctor.java:12: error: patient has private access in Doctor
          System.out.println("Patients  : " + aDoctor.getPatient() + " are waiting for Doctor"
+aDoctor.patient);
                                                                    ^
1 error
```

## Inheritance:

It is the property of creating a child class from a father class. When a new class is generated from another class, the properties of the father class are inherited within the child class as per some conditions. For making understanding easier we have used the terms 'father' and 'child' but, in the technical world, these two terms are denoted by 'Super' and 'Sub'. A super class can extend sub class and a sub class can extend another sub class. Multiple inheritance is not supported in JAVA as it is supported in C++. In Java, sub classes can be created from only one super class but not from multiple super classes. Java has no options to inherit from multiple parent classes. In human life, characteristics of father and mother may be inherited within the characteristic list of the child, but in Java it is not true; subclass can be created from only one super class, so, child class get all characteristics of super class except private characteristics.

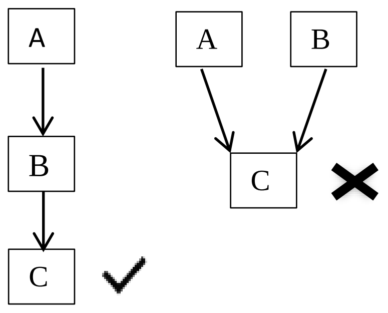

**Fig. 5.1.** graphical representation of inheritance of Java. Left-figure shows correct forms of multiple inheritances where the right-figure shows wrong inheritance in JAVA.

In the figure- it is shown that the Class 'A' can create another class 'B' and 'B' can create 'C'; advantages of the class 'C' is that it has access of all properties of 'B' and 'C' except private properties. This is the advantages of inheritance that the methods present within A and B are usable from C. But, the figure shown in right side, C can't inherit from A and B at a time from two super class; which is not allowed in JAVA. A child class or a derived class is created by keyword 'extends' as:

```
class B extends A
{
  ……
 ……
}
```

Here, a new class 'B' is created from superclass 'A' where this new class 'A' will inherit all properties of class 'A'. In the same way, another child class 'C' can be generated from class 'B' what is shown in left-figure of Fig.5.1 known as multiple inheritance.

In next example, a new class '**PetAnimal**' is created from the superclass '**Animal**'.

```
class PetAnimal extends Animal
{
….
….
}
```

The Superclass or Parent-class or Base class these are the three terms used to mean a **Base Class**, example is ‘**Animal**’ class shown above.

When is it required to create **Derived Class**? Answer is- if it is required to create a class which is similar to anyone of existing class, then, it is required to derive a new class from existing class. The new class may have extra features or properties than to Base class.

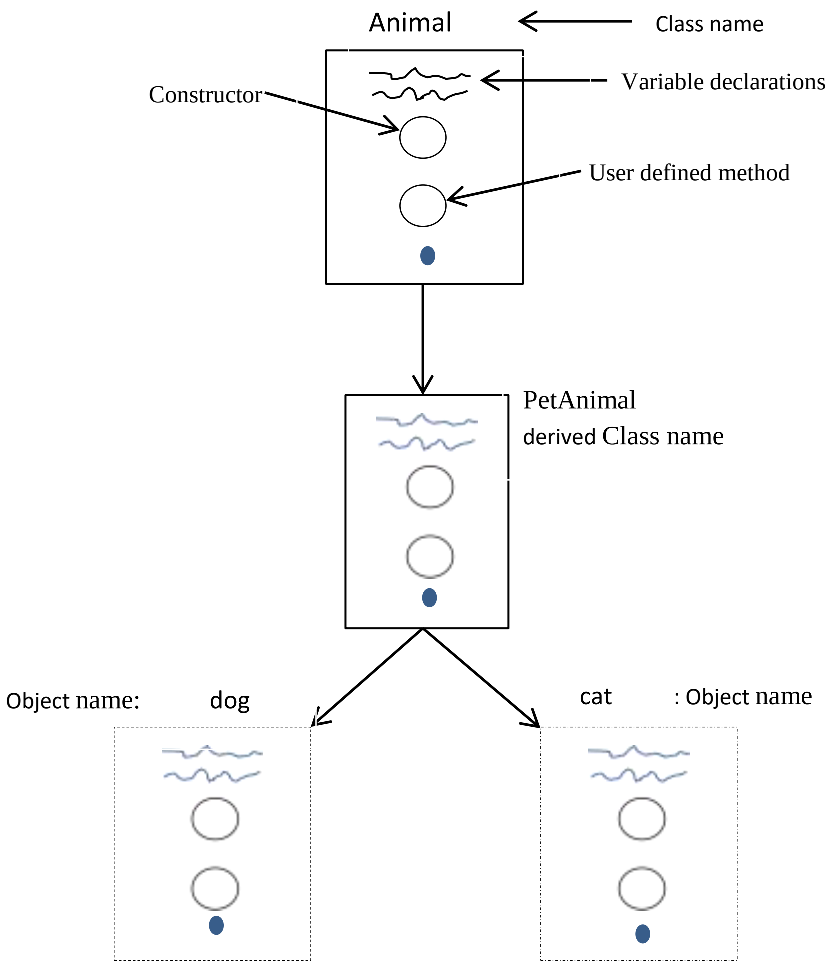

**Fig. 5.2.** Concept of Inheritance of properties from parent Class. This figure shows how to create a new class from parent class and then how to create objects from derived class

**Problem:** wap to create a class ‘Animal’ with a protected variable and a public method; create child class ‘PetAnimal’ from ‘Animal’ and then show the properties of inheritance rules of private, protected and public variables/methods.

```
//PetAnimal.java: demonstration of Inheritance
class Animal
{
   //private String color1 = "Has no fixed color...";
   protected String color1 = "Has no fixed color...";
   public void color()
   {
            System.out.println(" Animal has no fixed color.");
   }
}
//creating another class 'PetAnimal'
class PetAnimal extends Animal{
  private String color2 = "May be white or black or mixed";    // Car attribute
  public static void main(String[] args)
  {
     PetAnimal dog = new PetAnimal();
     PetAnimal cat = new PetAnimal(); // not used cat object here.
      System.out.println("Test of inheritance...");
     dog.color(); //calling public method
     System.out.println(dog.color1); //calling protected variable color1
     System.out.println(dog.color2);//calling private variable of same class
  }
}
```

```
OUTPUT:
Test of inheritance...
Animal has no fixed color.
Has no fixed color...
 May be white or black or mixed
```

**Discussion:** here one inherited class *PetAnimal* is created from the super class *Animal*. Sub class gets all properties inherited from the super class as: variable *color1* and the method *color()*, these are the two properties declared within *Animal* class which are accessible from the object of child. Advantage of inheritance is that the sub class need not to create new properties like: 'color1' and color( ) again as these properties are inherited to the subclass and it can use those properties as it's own. (It is like social life- as properties of Grand Father and Father as example Land, Home, Gold etc. are the properties of child/grandson also).

**Rules of Inheritance:** if any variable or method is declared as public or protected then it is inherited as usual i.e. public will be inherited as public, protected will be inherited as protected. Public method or variable is accessible from other package also; protected method or variable is accessible within the same package and same class. But, private property of parent class will not be inherited to the child class. Inherited class (object) has no access over the private property of parent class. As example, if we declare the variable as:

```
  private String color1 = "Has no fixed color...";
```

Then, ‘color1’ will not be inherited within child and also not accessible from the object of the subclass. To test this theory, in the above program, if this line is activated by deleting ‘//’ signs and blocking the next line as:

```
   //protected String color1 = "Has no fixed color...";
```

then compile/run the program it will show error as:

“*petAnimal.java:18: error: color1 has private access in Animal*”

```
    System.out.println(dog.color1); //from parent class
```

So, the theory we have learnt that ‘the private property of super class does not inherit to the sub class’ is also proved by this example.

```
//RoomVol.java: Class inheritance
// decision making statements if..else
class room
{
            int ll, bb;
            room(int x, int y)
            {
                        ll=x;      //length
                        bb=y;     //breadth
            }
            int area()
            {
                        return ll * bb;     //area
            }
}
// new class is inherited from another class room.
class mainroom extends room
{
            int ht;
            mainroom(int x, int y, int z)
            {
                        super(x, y);
                        ht=z;
            }
            int volume()
                        {
                                    return(ll * bb * ht);
                        }
}

class RoomVol
{
            public static void main(String args[])
            {
                        mainroom r1= new mainroom(10, 12, 8);
                        int ar=r1.area();
                        int vl=r1.volume();
                        System.out.println(" Room area  = " + ar);
                        System.out.println("Volume of the room = " + vl);
```

```
        }

}
```

```
OUTPUT
Room area  = 120
Volume of the room = 960
```

**Discussion:** in this example, first, a class '*room*' is created and within that room a constructor '*room()*' is created. It is noted that constructor '*room()*' has no data type; it assigns some values only. Within this class '*room*' another method is constructed by name 'area()' as integer type return data; it returns integer type of value after computing the line as: "*return ll * bb;*" . The class '*room*' has two main methods – one is 'room()' as constructor and another is '*area()*' having return type integer. The method is of integer data type meaning that the result will be computed by this method will be within the range of integer data. If any variable is assigned to hold that result that should be integer type also.

Next, another class '*mainroom*' is created from the class '*room*'. This way of creating a class is, extending from another class, known as creating child class from father class. Class '*mainroom*' has created a constructor by name '*mainroom*' having no datatype. When this class will be invoked from any instances of the class/object, this constructor '*mainroom*' will be executed automatically and then other jobs will be executed. Class '*mainroom*' has a method by name '*volume*' which is of integer datatype and return the volume of the room. Any numbers of methods or variables can be declared within the class.
Then, the class '*RoomVol*' is created which contains 'main()' method. The below line-

*mainroom r1= new mainroom(10, 12, 8);*

is the combination of two statements like:

*mainroom r1; // an object is created by name r1*
*r1 = new mainroom();// an instance r1 is created and memory is allocated as same size of 'mainroom' class*
*int ar =r1.area(); // variable 'ar' is assigned with the value of r1.area()*

Careful observation to the above program will show that class 'mainroom' is the child of super class 'room' and has no method 'area()' which is a method of super class 'room'; here, it is required to mention is that properties of super class is inherited within the sub/child class. So, when an object 'r1' is created from 'mainroom', that object contains all properties of the class of 'mainroom'. The instance of object has all access rights to the variables and methods of the class.

*The details of inheritance of properties are available in the referred books as mentioned in reference sections.

## Creating multilevel inheritance:

JAVA does not support multilevel inheritance, but there is an alternate way of accessing properties of multiple classes through the keyword '*implements*'. Through interfaces with an existing class (name as interface), properties of dual super classes can be accessed using *implements* keyword as the example shown below:

```
public class PetAnimal extends Animal implements WildAnimal
{
   //work with methods of Animal
....
....
  //work with interface of WildAnimal
    public void food()   //method of PetAnimal
    {
       //input weekly food amount
     }
}
```

**Discussions:** in this example, a new class '*PetAnimal*' is created from super class '*Animal*' again this new class inherits properties from the interface '*wildAnimal*'. An interface is like a class which is created by the keyword '*interface*' and generally declares some variables or methods within it but methods are not constructed within the interface. An interface is connected with a class by keyword '*implements*' as shown in the example of above. An existing class also can interface with an interface like '*WildAnimal*'.

## Multiple Choice Q & A

1. What is the role of this operator: <<
   a. To scroll one character to left
   b. To shift one bit to right side
   c. To shift one bit to left side
   d. None of the above.

2. When this symbols are used : [ ]
   a. For declaring arrays
   b. For assigning arrays
   c. For giving quotation marks at start and at ends of any string
   d. For all of the above.

3. In a byte type of data( of 8 bits), maximum positive number can be store is:
   a. 255
   b. -256
   c. -32768
   d. 32767
4. In programming, which package is imported by default ?
   a. java.io.DataInputStream
   b. java.lang
   c. java.util
   d. java.awt

5. fillOval(int,int,int,int) is a method of which Class?
   a. Thread
   b. String
   c. Graphics
   d. JFrame

6. What is the best answer of this statement: final double width = 300; ?
   a. Variable width is of double data type
   b. Variable width is of declared as non-overloading by final identifier.
   c. Data type of width is double and can't be declared in other classes or method as it is final and value
   300 is assigned within width.
   d. Data type is double of width, identifier is final meaning can't be overloaded.

7. If these two statements are given: Scanner in; String name = in.next(); and from keyboard 'Amal
   Kumar Khan' is inputted, then what is the contents of variable name?
   a. Amal Kumar Khan
   b. Amal Kumar
   c. Khan
   d. Amal

8. If these two statements are given as: **int[] num; num = new int[10];** then, which one is the best
answer?

a. 10 arrays are created
b. 10 arrays are created as: num[0] ….num[9]
c. 10 integers are created by name int[10]
d. Both 10 num and 10 int are created

9. Which one the best answer ?

a. Constructor has no data type
b. Constructor is called when object is initiated.
c. Name of Class and the Constructor should be same
d. Above all are correct

10. This statement : DataInputStream din= new DataInputStream(System.in); has the same effect if we write it in which way?

a. DataInputStream din=System.in;
b. DataInputStream din; din=new DataInputStream(System.in);
c. DataInputStream din new system.in;
d. DataInputStream din DataInputStream(System.in);

Ans: 1 ( c), 2(a), 3(a), 4(b), 5(c), 6(c), 7(d), 8(b), 9(d), 10(b)

Link of PPT of this book and all source codes:

```
https://drive.google.com/drive/folders/1bMxMCaqPe0W35COAdn_-BrnV_uzQ-tNO?usp=sharing
```

(Note: to get access, copy and paste this link to your browser)